

FIG. 1

200
↙

hotelchain(chainid, companyname, hqstate)
metroarea(metroid, metroname)
hotel(hotelid, hotelname, starrating, chain id
metro id, state id, city, pool, gym)
guestroom(r id, rhotel id, roomnumber, type, rackrate)
confroom(c id, chotel id, croomnumber, capacity, rackrate)
availability(a id, a r id, startdate, enddate, price)

FIG. 2

300
↙

```
<metro name="NYCMetroNJ">  
  <hotel name="Hyton" pool="Y" gym="N">  
    <confstat capacity="3200"/>  
    <confroom name="Green Flamingo" capacity="240"/> ...  
    ...  
  </hotel>  
  <hotel> ... </hotel> ...  
</metro>
```

FIG. 3

400
↙

```
<xsl:template match="pattern" mode="integer">
  <Result>
    <xsl:apply-templates select="expression"/>
  </Result>
</xsl:template>
```

FIG. 4

```

<xsl:template match="/" >                                (R1)
  <HTML>
    <HEAD></HEAD>
    <BODY>
      <xsl:apply-templates select="metro"/>
    </BODY>
  </HTML>
</xsl:template>
<xsl:template match="metro">                                (R2)
  <result_metro>
    <A></A>
    <xsl:apply-templates
      select="hotel/confstat"/>
  </result_metro>
</xsl:template>
<xsl:template match="confstat">                              (R3)
  <result_confstat>
    <B></B>
    <xsl:apply-templates
      select="../hotel_available/../confroom"/>
  </result_confstat>
</xsl:template>

<xsl:template match="metro/hotel/confroom">                 (R4)
  <xsl:value-of select="."/>
</xsl:template>

```

FIG. 5

Function $PROCESS(x, d_{con}, mode)$

Input: XSLT stylesheet as x , Context document node as d_{con} ,

Desired mode of the matched rule as $mode$

Output: XML document fragment as $result$

```

1:  $result \leftarrow empty$ 
2: for  $r_i \in x$  in decreasing order of  $priority(r_i)$  do
3:   if  $mode(r_i) = mode$  and  $MATCH(d_{con}, r_i)$  then
4:      $result \leftarrow output(r_i)$ 
5:     for  $a_j \in apply(r_i)$  do
6:        $D_{new\_con} \leftarrow SELECT(d_{con}, a_j)$ 
7:        $sub\_result \leftarrow empty$ 
8:       for  $d_{new\_con} \in D_{new\_con}$  do
9:          $sub\_result \leftarrow concatenate(sub\_result,$ 
            $PROCESS(x, d_{new\_con}, mode(a_j)))$ 
10:      Replace  $a_j$  in  $result$  with  $sub\_result$ 
11: return  $result$ 

```

FIG. 6

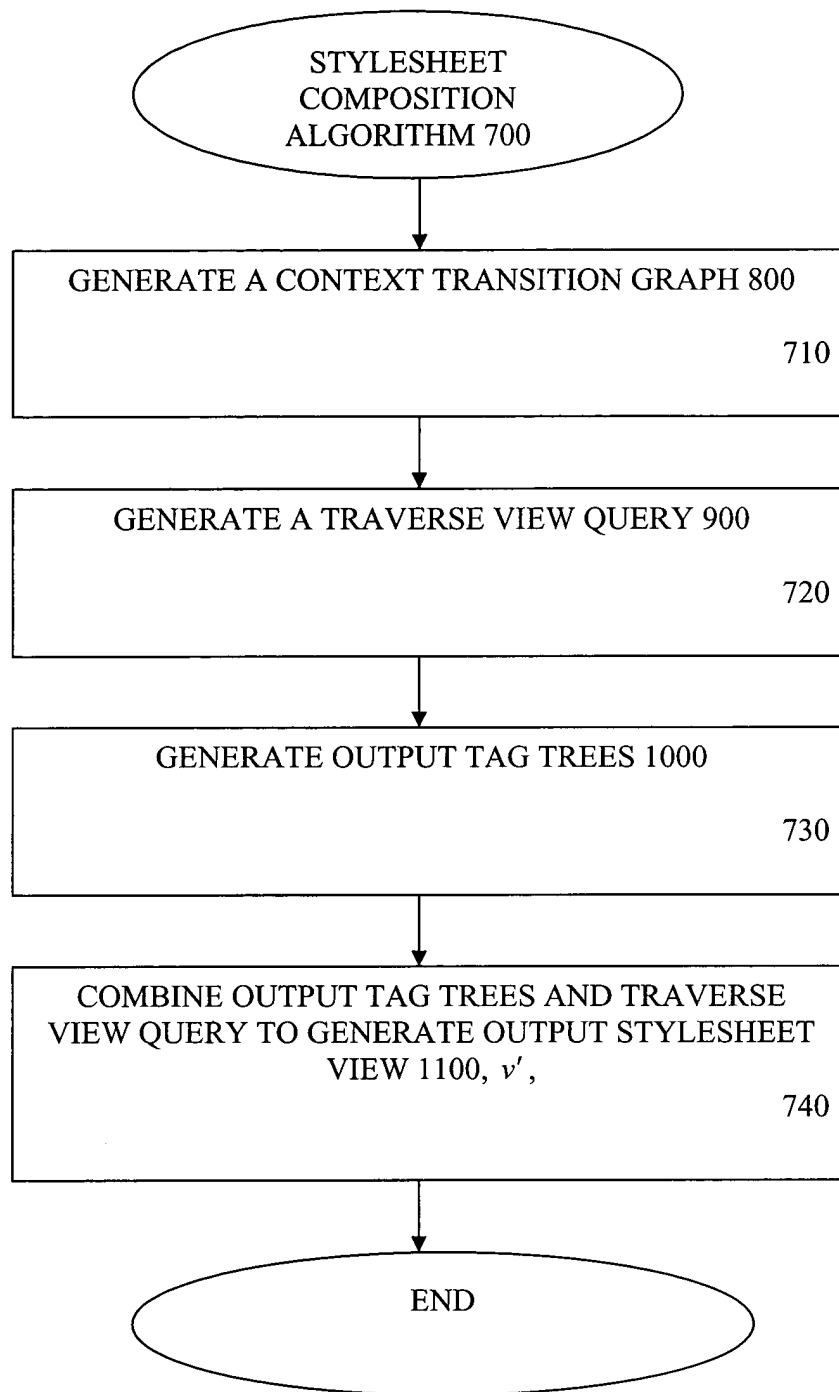


FIG. 7

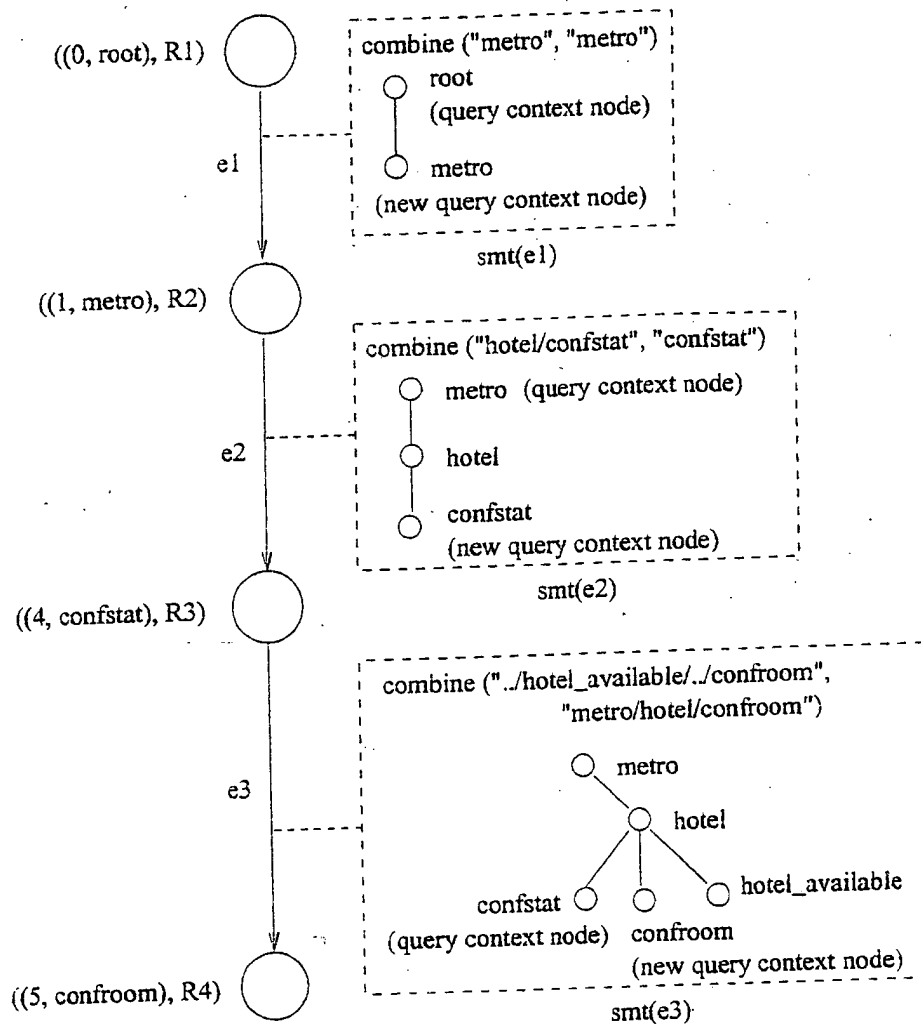
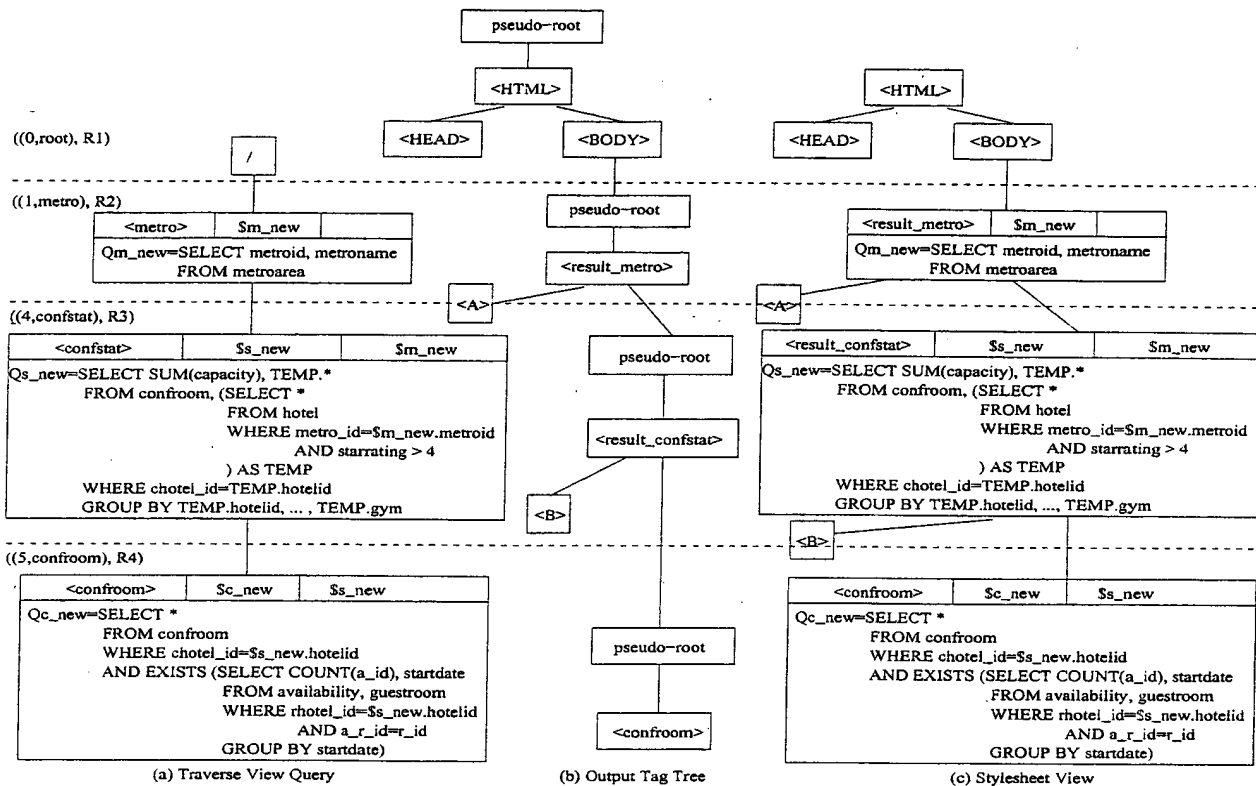


FIG. 8



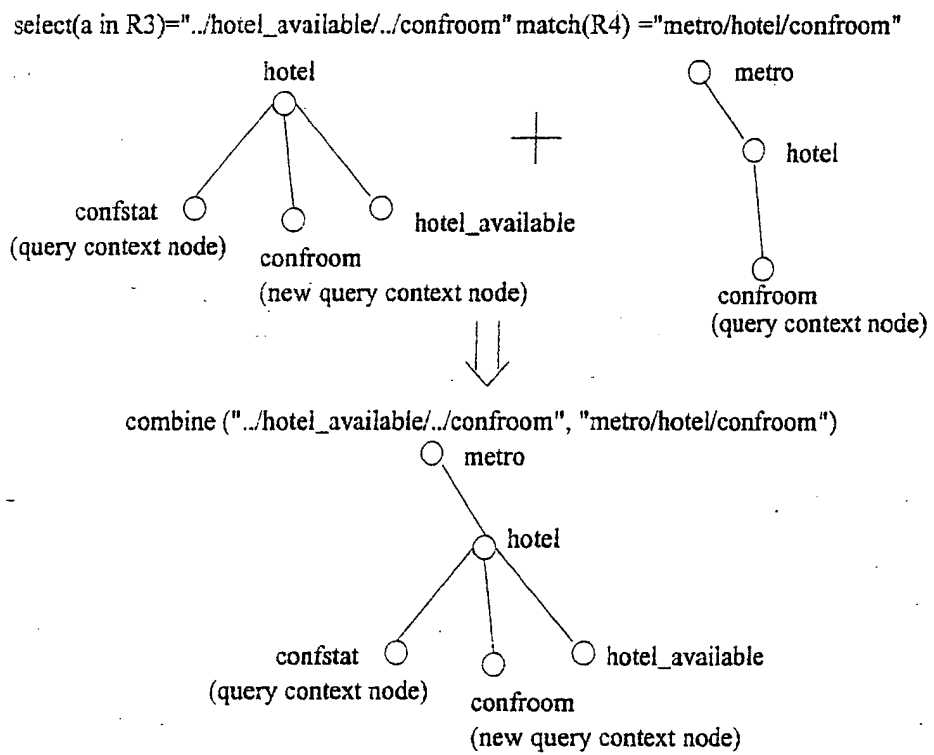


FIG. 12

Procedure *Compose*(v, x)

Input: v : original schema-tree view query; x : XSLT stylesheet

Output: stylesheet view

- 1: ctg : a Context Transition Graph
- 2: tvq : a Traverse View Query
- 3: $ottree$: an Output Template Tree
- 4: **for** $n \in v$ **do**
- 5: **for** $r \in x$ **do**
- 6: **if** $MATCHQ(n, r) \neq \text{NULL}$ **then**
- 7: add (n, r) to ctg
- 8: **for** $(n_1, r_1) \in ctg$ **do**
- 9: **for** $(n_2, r_2) \in ctg$ **do**
- 10: **for** $a \in \text{apply}(r_1)$ **do**
- 11: $t \leftarrow SELECTQ(n_1, a, n_2), p \leftarrow MATCHQ(n_2, r_2)$
- 12: **if** $t \neq \text{NULL}$ **and** $\text{mode}(a) = \text{mode}(r_2)$ **then**
- 13: add an edge $e = ((n_1, r_1), (n_2, r_2), a)$ to ctg
- 14: $smt(e) \leftarrow COMBINE(t, p)$
- 15: (repeatedly) Delete all nodes without incoming edge, except $(root, r)$
- 16: $tvq \leftarrow ctg \setminus \{(copy)\}$, $bvmap(\text{root of } tvq) \leftarrow \text{empty}$
- 17: (repeatedly) Duplicate nodes with multiple incoming edges, splitting incoming edges and copying outgoing edges
- 18: replace binding variables in tvq with new, unique binding variables
- 19: **for** $e = (w_1 = (n_1, r_1), w_2 = (n_2, r_2), a)$ in edges of ctg **do**
- 20: $(Q_{bv(w_2)}, bvmap(w_2)) \leftarrow$
 $UNBIND(smt(e), n_1, n_2, bv(w_2), bvmap(w_1))$
- 21: **for** all binding variables bv referenced in $Q_{bv(w_2)}$ **do**
- 22: rename bv as $bvmap(w_2).get(bv)$
- 23: **for** $w = (n, r) \in tvq$ **do**
- 24: $ott(w) = GENERATE_OTT(n, r)$
- 25: $ottree \leftarrow ott(w) \forall w \in tvq$ {initially a forest}
- 26: **for** $e = (w_1, w_2, a)$ in edges of tvq **do**
- 27: $a' \leftarrow$ the "apply-template" node in $ott(w_1)$ which is a copy of a
- 28: replace a' with an edge from $\text{parent}(a')$ to $\text{root}(ott(w_2))$
- 29: **for** $w = (n, r) \in tvq$ **do**
- 30: $bv(\text{root}(ott(w))) \leftarrow bv(w)$
- 31: $Q_{bv(\text{root}(ott(w)))} \leftarrow Q_{bv(w)}$
- 32: Remove the topmost "pseudo-root" node in $ottree$
- 33: **while** there is any "pseudo-root" node pr left **do**
- 34: **for** each child node c of pr **do**
- 35: add edge $e = (\text{parent}(pr), c)$
- 36: **if** $Q_{bv(c)}$ is empty **then**
- 37: $bv(c) \leftarrow bv(pr), Q_{bv(c)} \leftarrow Q_{bv(pr)}$
- 38: **else**
- 39: $Q_{bv(c)} \leftarrow UNBIND(\text{parent}(pr), c)$
- 40: add the *SELECT* columns of $Q_{bv(pr)}$ to $Q_{bv(c)}$
- 41: change " $bv(pr)$ " as " $bv(c)$ " in the tag queries of c 's descendants
- 42: remove edge $e = (\text{parent}(pr), pr)$ and node pr
- 43: **return** $ottree$ {new stylesheet view}

FIG. 13

Function $UNBIND(m, n)$

Input: Query context node as m , Target node to unbind as n

Output: Unbound query for n as q

- 1: $n_j \leftarrow$ the lowest common ancestor of m and n
- 2: $s_j \leftarrow bv(n_j)$
- 3: $child_n(n_j) \leftarrow$ child node of n_j along the path from n_j to n
- 4: $s_{j-1} \leftarrow bv(child_n(n_j))$
- 5: $q \leftarrow Q_{bv(n)}^{s_1, s_2, \dots, s_{j-1}}(s_j, \dots, s'_k)$, which is the unbound tag query of n , $Q_{bv(n)}(s_1, s_2, \dots, s_k)$
- 6: **return** q

FIG. 14

Function $NEST(p, p')$

Input: Node as p , Child node as p'

Output: Tag query for p after nesting as q

- 1: $\bar{q} \leftarrow Q_{bv(p)}$
- 2: **for** each child node c of p , except p' **do**
- 3: $q_c \leftarrow NEST(c, NULL)$
- 4: add $EXISTS$ q_c in $WHERE$ clause of \bar{q}
- 5: **return** q

FIG. 15

- 1: {Replace line 5 of Figure 10}
- 2: $P \leftarrow$ {nodes along the path from $child_n(n_j)$ to n }
- 3: **for** node $p \in P$ **do**
- 4: $p' \leftarrow$ child of $p \in P$, otherwise $NULL$
- 5: $\Theta_{bv(p)} \leftarrow NEST(p, p')$
- 6: $q \leftarrow \Theta_{bv(n)}^{s_1, s_2, \dots, s_{j-1}}(s_j, \dots, s'_k)$, which is the unbound and nested tag query of n , $\Theta_{bv(n)}(s_1, s_2, \dots, s_k)$, and decorrelation of $bv(s_i)$ is done by Θ_{s_i} .

FIG. 16

Function *UNBIND*(*smt*, *m*, *n*, *bv'*, *bvmap*)

Input: Select-match subtree as *smt*, Query context node of *smt* as *m*, New query context node of *smt* as *n*, New binding variable as *bv'*, Binding variable map as *bvmap*.

Output: Unbound query for *smt* as *q*, New binding variable map as *bvmap'*.

- 1: $q \leftarrow \text{UNBIND}(m, n)$
- 2: $n_j \leftarrow$ the lowest common ancestor of *m* and *n*
- 3: $\text{child}_n(n_j) \leftarrow$ child node of n_j along the path from n_j to *n*
- 4: $R \leftarrow \{\text{nodes along the path from } \text{child}_n(n_j) \text{ to } n\}$
- 5: **for** node $p \in R$ **do**
- 6: add the *SELECT* columns of $Q_{bv(p)}$ to *q*
- 7: $P \leftarrow \{\text{nodes along the path from root of } smt \text{ to } m\}$
- 8: **for** node $p \in P$ **do**
- 9: **for** each child node *c* of *p*, such that $c \notin P$ and $c \notin R$ **do**
- 10: $q_c \leftarrow \text{NEST}(c, \text{NULL})$
- 11: add *EXISTS* q_c in *WHERE* clause of *q*
- 12: $bvmap' \leftarrow bvmap$
- 13: **for** node $p \in R$ **do**
- 14: $bvmap'.insert(bv(p), bv')$
- 15: $\text{child}_m(n_j) \leftarrow$ child node of n_j along the path from n_j to *m*
- 16: $S \leftarrow \{\text{nodes along the path from } \text{child}_m(n_j) \text{ to } m\}$
- 17: **for** node $s \in S$ **do**
- 18: $bvmap'.remove(bv(s))$
- 19: **return** (*q*, *bvmap'*)

FIG. 17

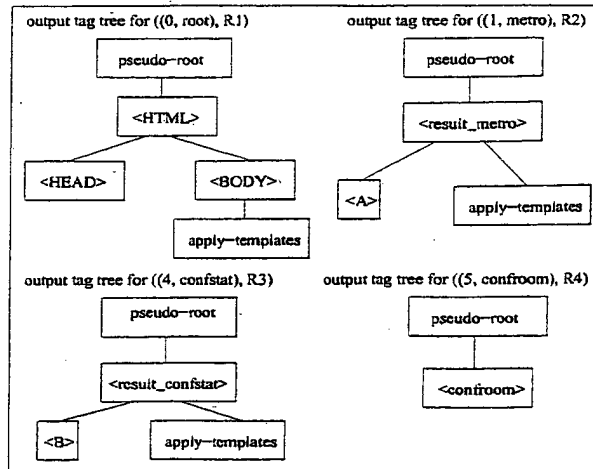


FIG. 18

(R1), (R3), and (R4) are the same as Figure 4.

```
<xsl:template match="metro">                                (R2)
  <xsl:apply-templates select="hotel/confstat"/>
</xsl:template>
```

FIG. 19

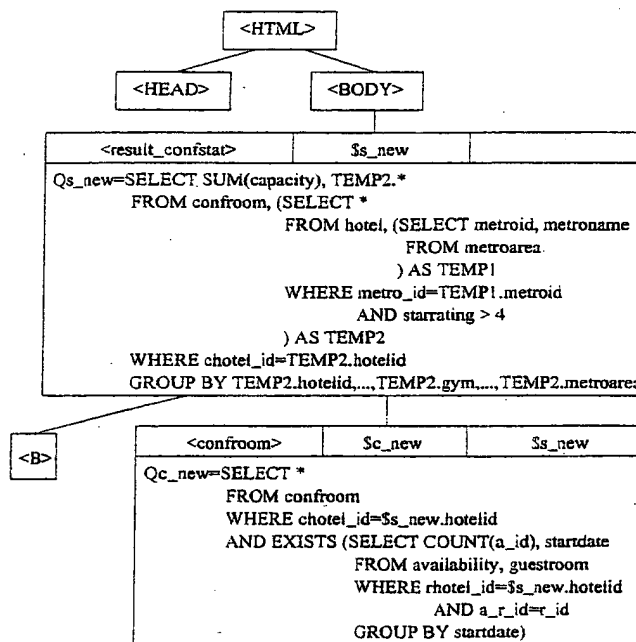


FIG. 20

(R1) and (R2) are the same as Figure 4.

```

<xsl:template match="confstat">                                (R3)
  <result_confstat>
    <B/>
    <xsl:apply-templates select=".[@sum<200]/
      ../hotel_available/../../confroom
      [../../confstat[@sum>100]][@capacity>250]"/>
  </result_confstat>
</xsl:template>

<xsl:template match="metro[@metroname=
  "chicago"]/hotel/confroom">                                (R4)
  <xsl:value-of select="."/>
</xsl:template>

```

FIG. 21

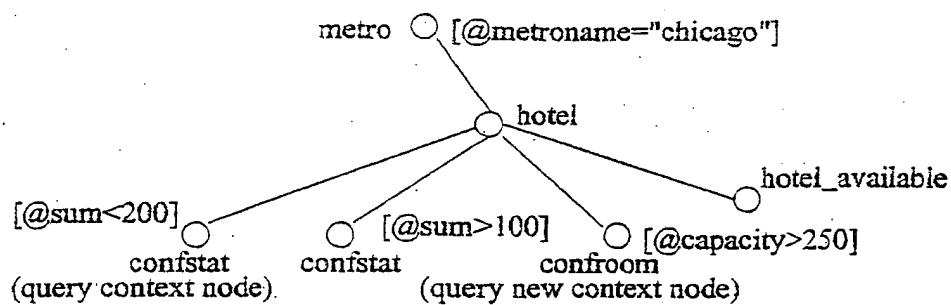


FIG. 22

{Insert the following
 (a) after line 8 of Figure 13
 (b) after line 1 of Figure 11}
 $e_p \leftarrow$ predicate using $tag(p)$
 add e_p in *WHERE* clause of q

FIG. 23

```

SELECT * FROM confroom
WHERE chotel_id=$s_new.hotelid
  AND capacity > 250
  AND $s_new.SUM_capacity<200
  AND $s_new.metroname=''chicago''
  AND EXISTS (SELECT SUM(capacity)
              FROM confroom,
              WHERE chotel_id=$s_new.hotelid
              HAVING SUM(capacity)>100)
  AND EXISTS (SELECT COUNT(a_id), startdate
              FROM availability, guestroom
              WHERE rhotel_id=$s_new.hotelid
              AND a_r_id=r_id
              GROUP BY startdate)

```

FIG. 24

```

<xsl:template match="pattern" mode="m">
  <xsl:if test="expression">
    template body
  </xsl:if>
</xsl:template>

```

(a)

```

<xsl:template match="pattern" mode="m">
  <xsl:apply-templates select=". [expression]" mode="mnew"/>
</xsl:template>

<xsl:template match="nodename" mode="mnew">
  template body
</xsl:template>

```

(b)

FIG. 25

```

<xsl:template match="pattern" mode="m">
  <xsl:choose>
    <xsl:when test="e1">b1</xsl:when>
    <xsl:when test="e2">b2</xsl:when>
    <xsl:otherwise>b3</xsl:otherwise>
  </xsl:choose>
</xsl:template>

```

(a)

```

<xsl:template match="pattern" mode="m">
  <xsl:apply-templates
    select=". [e1]" mode="mnew1"/>
  <xsl:apply-templates
    select=". [not (e1) and e2]"
    mode="mnew2"/>
  <xsl:apply-templates
    select=". [not (e1) and not (e2)]"
    mode="mnew3"/>
</xsl:template>

```

```

<xsl:template match="nodename"
  mode="mnew1">b1</xsl:template>
<xsl:template match="nodename"
  mode="mnew2">b2</xsl:template>
<xsl:template match="nodename"
  mode="mnew3">b3</xsl:template>

```

(b)

FIG. 26


```

<xsl:template match="pattern" mode="m">
  <xsl:value-of select="expression"/>
</xsl:template>

```

(a)

```

<xsl:template match="pattern" mode="m">
  <xsl:apply-templates
    select="path expression" mode="mnew"/>
</xsl:template>

```

```

<xsl:template match="nodename[predicate]"
  mode="mnew">
  <xsl:value-of select="."/>
</xsl:template>

```

(b)

FIG. 27

```

<xsl:template match="pattern 1" mode="m">
  template body 1
</xsl:template>

```

```

<xsl:template match="pattern 2" mode="m">
  template body 2
</xsl:template>

```

(a)

```

<xsl:template match="pattern 1" mode="m1">
  template body 1
</xsl:template>

```

```

<xsl:template match="pattern 2" mode="m">
  <xsl:choose>
    <xsl:when test="expression 1">
      <xsl:apply-templates select="."
        mode="m1"/>
    </xsl:when>
    <xsl:otherwise>
      template body 2
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

```

(b)

FIG. 28

```

<xsl:template match="/metro">                                (R1)
  <xsl:param name="idx" select="10"/>
  <result_metro>
    <xsl:apply-templates
      select="hotel/hotel_available[@count>10]
        /metro_available[@count<$idx]">
      <xsl:with-param name="idx"
        select="$idx"/>
    </xsl:apply-templates>
  </result_metro>
</xsl:template>

<xsl:template match="metro_available">                        (R2)
  <xsl:param name="idx"/>
  <xsl:choose>
    <xsl:when test="$idx<=1">
      <xsl:value-of select="."/>
    </xsl:when>
    <xsl:otherwise>
      <result_metroavail>
        <xsl:apply-templates
          select="self::[@count>50]/../..../..">
          <xsl:with-param name="idx"
            select="$idx-1"/>
        </xsl:apply-templates>
      <result_metroavail>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

```

FIG. 29

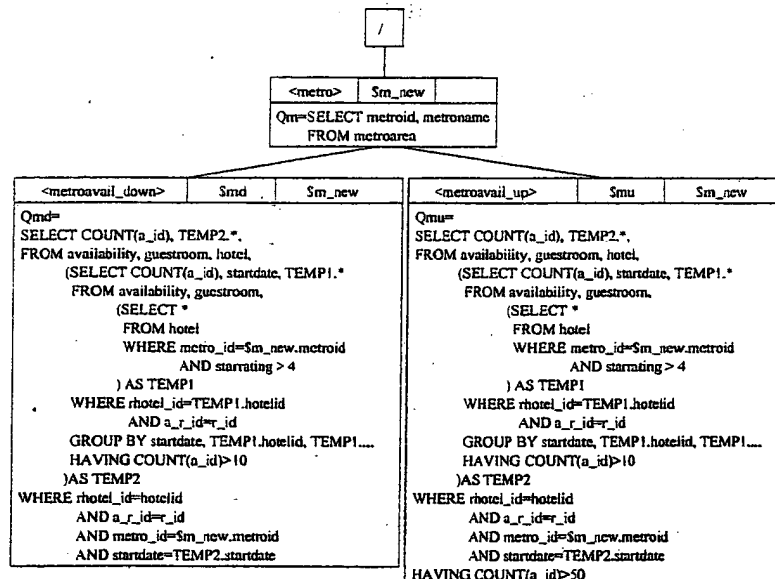


FIG. 30

```

<xsl:template match="/metro">                                     (R1')
  <xsl:param name="idx" select="10"/>
  <result_metro>
    <xsl:apply-templates
      select="metroavail_down[@count<$idx]">
      <xsl:with-param name="idx"
        select="$idx"/>
    </xsl:apply-templates>
  </result_metro>
</xsl:template>

```

```

<xsl:template match="metroavail_down">                             (R2')
  <xsl:param name="idx"/>
  <xsl:choose>
    <xsl:when test="$idx<=1">
      <xsl:value-of select="."/>
    </xsl:when>
    <xsl:otherwise>
      <result_metroavail>
        <xsl:apply-templates
          select="../metroavail_up">
          <xsl:with-param
            name="idx" select="$idx-1"/>
        </xsl:apply-templates>
      <result_metroavail>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

```

```

<xsl:template match="metroavail_up">                               (R3')
  <xsl:param name="idx"/>
  <result_metro>
    <xsl:apply-templates
      select="../metroavail_down[@count<$idx]">
      <xsl:with-param
        name="idx" select="$idx"/>
    </xsl:apply-templates>
  <result_metro>
</xsl:template>

```

FIG. 31